Lambda Types on the Lambda Calculus with Abbreviations

Ferruccio Guidi
Department of Computer Science
University of Bologna

Abstract for an informal presentation at CIE 2007

$\lambda\delta$ [2] is a typed λ-calculus that pursues the reuse of the term constructions both at the level of types and at the level of contexts, while enjoying the most common meta-theoretical properties [1]. Notice that the distinction we are making here between terms and types is borrowed from the simply typed λ-calculus $\lambda{\rightarrow}$. The development of $\lambda\delta$ is intended as the first step towards the verification of the following conjecture: the adoption of a single set of constructions for terms, types, and contexts (i.e. the "contexts as types as terms" paradigm) is compatible with the presence of a desirable meta-theory. $\lambda\delta$ features the term constructions of Church $\lambda{\rightarrow}$ plus sorts, abbreviations and type casts. Sorts are necessary to build closed terms, abbreviations (i.e. let expressions) are essential in the proofs of some meta-theoretical properties when the "types as terms" paradigm is adopted and are practically unavoidable in mathematics and computer science, while type casts play an important role in connection to canonical typing and are borrowed from realistic type checker implementations. $\lambda\delta$ realizes the "types as terms" paradigm in full and features an uniform typing policy for all binders, i.e. for each binder $\flat$, a $\flat$-abstraction is typed with a $\flat$-abstraction by means of a uniform rule. As a consequence $\lambda\delta$ is a λ-typed λ-calculus but it differs from the Automath-related calculi [4] in that they do not provide for an abbreviation construction at the level of terms. Moreover $\lambda\delta$ features a type hierarchy with a potentially infinite number of levels both above and below any reference point. $\lambda\delta$ realizes the "contexts as terms" paradigm only partially since a context is always a special case of a term but the converse is not true in general (the author is studying an extension of $\lambda\delta$ meant to solve this problem). The reduction steps of $\lambda\delta$ include (but are not limited to) β-contraction, δ-expansion and ζ-contraction. On the other hand η-contraction is not included. The meta-theory of $\lambda\delta$ includes the meta-theory of Church $\lambda{\rightarrow}$ (i.e. confluence of reduction, subject reduction, strong normalisation and decidability of type inference) and, in addition, the correctness of types and the uniqueness of types can also be proved. $\lambda\delta$ shares with Church $\lambda{\rightarrow}$ the subset of typable terms but in the "propositions as types" perspective it can encode the implicative fragment of predicative logic without quantifiers because dependent types are allowed. Moreover the λ-abstraction is predicative in the sense that $C \vdash \lambda x{:}V.T : V$ never holds so the calculus can serve as a formal specification language for the type theories, like mTT [3], that require to be expressed in a predicative foundation.

References

[1] H.P. Barendregt. Lambda Calculi with Types. *Osborne Handbooks of Logic in Computer Science*, 2:117–309, 1993.

[2] F. Guidi. Lambda-Types on the Lambda-Calculus with Abbreviations, Nov 2006. http://arxiv.org/abs/cs/0611040.

[3] M.E. Maietti and G. Sambin. Towards a minimalist foundation for constructive mathematics. In L. Crosilla and P. Schuster, editors, *From Sets and Types to Topology and Analysis: Practicable Foundations for Constructive Mathematics*. Oxford University Press, Oxford, 2005. Forthcoming.

[4] D.T. van Daalen. The language theory of Automath. Ph.d. thesis, Eindhoven University of technology, 1980.